Simulating Language: Lab 2 Worksheet

If you haven't already done so, create a directory in your home area (on the M: drive) for the Simulating Language course. Download the file `signalling1.py` from the website and save it to this directory.

To open this file in IDLE, choose **File-Open** from the menu, and find the file in the directory you have just created.

1. Simple Innate Signalling

At the start of the file are some detailed comments explaining what the code is supposed to do. Comments in programming are very useful, both to you and those who look at your code later; they should be explanatory, clear, concise and accurate, but first and foremost they should be **there**. Comments are enclosed with triple-quotes, normally set off on their own lines. Single-line comments begin with a hash (**#**), and continue until the end of the line.

As you can see, the function `ca_monte` measures and returns the level of communicative accuracy between a production system and a reception system. The comments give brief details about the data structures, and examples of how the program should be used. In this case, we see that the signalling systems are stored as lists of lists of association weights; this list of lists structure can be thought of as a matrix with meanings on the rows and signals on the columns (for production matrices) or signals on the rows and meanings on the columns (for reception matrices).

signalling1.py

```
"""
Simple innate signalling simulation

ca_monte returns communicative accuracy between a speaker (or producer) system
and a hearer (or receiver) system using monte carlo simulation.

Systems are expressed as a list of lists of association weights. Matrix rows in the
speaker system are meanings, columns are signals. In the hearer system, matrix
rows are signals, columns are meanings.

Production and reception are winner-take-all.

Usage example (note: I have presented the speaker and hearer systems row-by-row
to make the matrix structure clearer: there is no need to do this when using
the code, unless you feel it helps.

a_speaker_system = [[1, 0, 0],
                    [0, 1, 0],
                    [0, 1, 1]]
a_hearer_system =  [[1, 0, 0],
                    [0, 1, 1],
                    [0, 0, 1]]
ca_monte(a_speaker_system, a_hearer_system, 100)

Returns a list of expected communicative success values, in a trial-by-trial
list (so the first element in the list gives the proportion of successful
communications after 1 trial, the second gives the proportion of successful
```

```
events after two trials etc), based on 100 evaluations of communication between
the specified speaker and hearer systems.  There are three meanings
and three signals, but the communication system as specified above contains
some homonymy (the second signal can be used for either the second or third
meaning) and synonymy (the third meaning can be expressed using either the
second or third signal).
"""


import random
import matplotlib.pyplot as plt

def wta(items):
    maxweight = max(items)
    candidates = []
    for i in range(len(items)):
        if items[i] == maxweight:
            candidates.append(i)
    return random.choice(candidates)

def communicate(speaker_system, hearer_system, meaning):
    speaker_signal = wta(speaker_system[meaning])
    hearer_meaning = wta(hearer_system[speaker_signal])
    if meaning == hearer_meaning:
        return 1
    else:
        return 0

def ca_monte(speaker_system, hearer_system, trials):
    total = 0.
    accumulator = []
    for n in range(trials):
        total += communicate(speaker_system, hearer_system,
                             random.randrange(len(speaker_system)))
        accumulator.append(total/(n+1))
    return accumulator
```

The program proper begins with a rather lengthy comment explaining what it does and how to use it, followed by two import commands which import the random and matplotlib libraries (so that we can generate and use random numbers and produce plots of results), and then defines the individual functions.

Make sure you understand what each function does. Look at the main function `ca_monte` first, then the function it calls (`communicate`), and so on until you have inspected each function separately. Can you see why the program has been divided into functions in the way it has? **If you cannot figure out what the code does** by yourself (and with our help) then download worksheet02_walkthrough from WebCT (you'll find it in the Lab worksheets folder): this gives detailed line-by-line comments, which might help.

To run the code, choose **Run-Run Module** from the menu. Create a production and reception matrix as shown in the comments, and calculate its communicative accuracy. When you are satisfied that you understand how the code works, answer the following questions. 1-3 should be completed by everyone. **Only attempt 4 and 5 if you are happy you have completed 1-3.**

1. How many trials should there be in the Monte Carlo simulation to work out communicative accuracy? Hint: answer this question empirically by plotting the results and comparing to what the "real" answer should be for various numbers of trials. You will want to use the plot function from matplotlib to do this - look back over the lab 1 worksheet for details.

2. How do synonymy and homonymy affect communicative accuracy? Create production and reception systems with different degrees of homonymy and synonymy to explore this. Note: you don't have to restrict yourself to systems with 3 meanings and 3 signals.

3. What alternatives to "winner-take-all" might there be in the model of production/ reception? What difference might this make? Would they be more or less realistic, or powerful as a model? Hint: how might you interpret weights as probabilities?

[4 and 5 are optional: only attempt these if you are satisfied you thoroughly understand 1-3]
4. How could you model production and reception using a single underlying matrix, rather than separate production and reception matrices? Is this kind of model better or worse than a model where we use separate matrices?

5. How would you go about calculating communicative accuracy exactly, i.e. rather than via Monte Carlo techniques?